# Project Objectives

This is a partner project that involves creating functions to combine and summarize data.

## Setting Things Up

### Creating the Repo

The first step is for the *first* group member (first member listed in the group) to create a github repo and link it with an RStudio project. Push up those changes so the .Rproj file exists. By using an R project you can stick to using relative file paths for everything. Next, go to the repo **settings** on github and add the *second* group member as a collaborator (you'll need their github account). The second group member then needs to accept the membership. This gives everyone access to push changes up to the repository. **All project work should be done within this repo.**

You will be generating a pdf, and uploading that PDF to Gradescope as we have done on other assignments.

### Collaboration Workflow

You and your partner should collaborate on the project. You should set up a list of tasks and timelines/milestones with flex time. I know some people tend to just try to do all the work for the project themselves. **If this is done, both people in the group will lose credit.** This should be a collaboration. We will be checking the commits on the repo to make sure both people are contributing substantially. **With this in mind, commit often!** Don't work a ton locally without pushing things ups. We want to see the work develop on github. Again, if we can't see the contributions via github, both members may lose credit.

Please reach out early if the agreed upon milestones/work are not being completed by your partners.

For collaborating, you have two options:

- Both work on the same (main) branch. If this is the case, your workflow may look like the following:
  - Each time you go to work on the project, you pull down any of the latest changes (`git pull`)
  - When you are satisfied with some contributions, add, commit, and push your work up to the main branch. (Remember version control is part of the system so you can always roll back changes.)
  - There may occasionally be merge conflicts that have to be dealt with. This can be done with the `Git` tab in RStudio. Let us know if you are having issues with conflicts that you can't resolve!
- You each work on your own branch (or one on the main branch and one on their own branch). Notes about that process:
  - A branch can be created on github.com or via the command line with `git checkout -b new_branch_name`. Any work you then do will be on that branch.
  - In order to push up the new branch and any changes to github, you can do your usual add and commit steps. The first time you do a push step, you have to add some extra code: `git push --set-upstream origin branch_name`
  - You can switch between branches with `git checkout branch_name`
  - You can merge in your branch's changes to the `main` branch by
    1. Checking out the main branch: `git checkout main`
    2. Merging your changes in: `git merge branch_name`
  - As with the other workflow, you may need to resolve merge conflicts. We're happy to help out if you get stuck on that!

## .qmd Format

There are points allocated to your .qmd format. Please see the instructions below.

Use second level headers to label each question. Label your questions appropriately (ex. Question 1). You are encouraged to name your questions descriptively (ex. Question 1: select columns).

All messages and warnings that come from librarying packages should be turned **off** using the appropriate code chunk option. All messages from loading in your data should be turned **off** as well. This will make your project report look more tidy.

## Overall Goal

Our goal is to write functions that will manipulate and process data sets that come in a certain form. We'll create a *generic* function to automatically plot the returned data. We'll write up a document via quarto (just like we did with the homework). You will also be asked to describe our thought process and give examples of using the functions. **You must have a narrative throughout the document.** This project is as much about explaining your thought process / what the code is doing as it is the code itself.

### Data

We'll use a number of `.csv` files that contain information from the census bureau. This data is a little older (2010). Our first goal will be to read in one of these `.csv` files and parse the data using functions we've written. Then we'll combine some parsed data and deal with it from there.

### Data Processing

**First Steps** First I'll just explain what you'll do to parse the data you read in. Then I'll give you requirements on how to parse it below. (I find it easier to first do all of the things below and then convert certain things into functions and what-not. Feel free to create functions as required first if you'd like.)

Read in one section of the data. This data is available at https://www4.stat.ncsu.edu/~online/datasets/EDU01a.csv.

1. Select only the following columns:
    - `Area_name` (rename as `area_name`)
    - `STCOU`
    - Any column that ends in "D"

Display the first 5 rows of your new data set to show that you created this correctly. Note: Do not save over your new data set with just the first 5 rows, simply just show the first 5 rows.

2. Convert the data into long format where each row has only one enrollment value for that `Area_name`. Display the first 5 rows of your new data set to show that you created this correctly.
3. One of the new columns should now correspond to the old column names that end with a "D". All columns in these census data files will have this similar format. The first three characters represent the survey with the next four representing the type of value you have from that survey. The last two digits prior to the "D" represent the year of the measurement. For more about the variables see the data information sheet).
    - Parse the string to pull out the year and convert the year into a **numeric** value such as `1997` or `2002`.

- Grab the first three characters and following four digits to create a new variable representing which measurement was grabbed.
  - Hint: Check out the `substr()` function from base R.

Save this data set as `long_updated`, and again show the first 5 rows.

4. Create two data sets

   - one data set that contains only non-county data
   - one data set that contains only county level data

   Note that all county measurements have the format "County Name, DD" where "DD" represents the state. This can be used to subset the data. I used the code `grep(pattern = ", \\w\\w", Area_name)` to get the indices corresponding to counties.

   For the county level data, add a `class` to the tibble called `county`. Similarly, add a `class` to the non-county data called `state`. This can be done by overwriting the `class()` you see on the object:

```
class(your_county_tibble) <- c("county", class(your_county_tibble))
```

Print the first 10 rows of each tibble by including `county_tibble` and `state_tibble` in your code chunk.

5. For the county level tibble, create a new variable that describes which state one of these county measurements corresponds to (the two digit abbreviation is fine, see `substr()`).
6. For the non-county level tibble, create a new variable called "division" corresponding to the state's classification of division here. If row corresponds to a non-state (i.e. `UNITED STATES`), return `ERROR` for the division. Hint: Use `%in%`and consider `if_else` or `case_when` logic.

**Requirements** Now we want to repeat the above process for the 2nd component of the data set. This is available at the link below and can be read in by modifying the `read_csv()` code given earlier

- https://www4.stat.ncsu.edu/~online/datasets/EDU01b.csv

Rather than copying and pasting a bunch of stuff and changing things here and there, we want to write functions that do the above pieces and one function that we can call to do it all!

- Write one function that does steps 1 & 2 above. Give an optional argument (that is it has a default value) that allows the user to specify the name of the column representing the value (enrollment for these data sets).
- Write another function that takes in the output of step 2 and does step 3 above.
- Write a function to do step 5
- Write a function to do step 6
- Write another function that takes in the output from step 3 and creates the two tibbles in step 4, calls the above two functions (to perform steps 5 and 6), and returns two final tibbles.

Now last thing, put it all into one function call! This is called creating a wrapper function. Create a function that takes in the URL of a `.csv` file in this format and the optional argument for the variable name, calls the functions you wrote above, and then returns the two tibbles.

```r
#here is the idea in case it isn't clear
my_wrapper <- function(url, default_var_name = "default of some kind"){
  result <- read_csv_code(...) |>
    function_for_step_1_2(...) |>
    function_for_step_3(...) |>
    function_for_steps4_5_6(...)
  #return final result
  }
```

```r
long_format_conversion <- function(df, value = "Enrollment Value") {
  selected_data <- df |>

# Select the respective columns
  select(Area_name, STCOU, ends_with("D")) |>

# Rename the Area_name variable
  rename(area_name = Area_name)

# Convert the data into long format
  long_data <- pivot_longer(selected_data, cols = ends_with("D"), names_to = "Survey", values_to = valu
  return(long_data)
}

survey_function <- function(long_data) {
  long_data_updated <- long_data |>

  # Extract the year from the 'Survey' column, covert to a number, and create a new 'Year' variable wit
  mutate(Year = as.numeric(substr(Survey, start = 8, stop = 9))) |>

  # Covert the two digit year into a four digit year and overwrite the 'Year' column
  mutate(Year = ifelse(Year > 25, Year + 1900, Year + 2000)) |>

  # Extract the first 7 digits from the 'Survey' column and create a new 'Measurement' variable with th
  mutate(Measurement = substr(Survey, start = 1, stop = 7))
  return(long_data_updated)
}

state_function <- function(county_tibble){
  new_county_tibble <- county_tibble |>
  mutate(State = substr(area_name, start = nchar(area_name)-1, stop = nchar(area_name)))
  return(new_county_tibble)
}

division_function <- function(noncounty_tibble) {
  noncounty_tibble_updated <- noncounty_tibble |>
    filter(area_name != "District of Columbia") |>
  mutate(Division = ifelse(area_name %in% c("CONNECTICUT", "MAINE", "MASSACHUSETTS", "NEW HAMPSHIRE", "
                           ifelse(area_name %in% c("NEW JERSEY", "NEW YORK", "PENNSYLVANIA"), "Division
                                  ifelse(area_name %in% c("ILLINOIS", "INDIANA", "MICHIGAN", "OHIO", "W
                                         ifelse(area_name %in% c("IOWA", "KANSAS", "MINNESOTA", "MISSOU
                                                ifelse(area_name %in% c("DELAWARE", "FLORIDA", "GEORGIA
                                                       ifelse(area_name %in% c("ALABAMA", "KENTUCKY", "
                                                              ifelse(area_name %in% c("ARKANSAS", "LOUI
```

```
                                                                    ifelse(area_name %in% c("ARIZONA",
                                                                           ifelse(area_name %in% c("AL
  return(noncounty_tibble_updated)
}

create_datasets <- function(long_data) {
  county_indices <- grep(pattern = ", \\w\\w", long_data$area_name)
  noncounty_tibble <- long_data[-county_indices, ]
  class(noncounty_tibble) <- c("state", class(noncounty_tibble))
  county_tibble <- long_data[county_indices, ]
  class(county_tibble) <- c("county", class(county_tibble))
  final_county_tibble <- state_function(county_tibble)
  final_noncounty_tibble <- division_function(noncounty_tibble)
  return(list(county = final_county_tibble, noncounty = final_noncounty_tibble))
}

my_wrapper <- function(url, value = "Enrollment Value"){
result <- read_csv(url) |>
long_format_conversion(value = value) |>
survey_function() |>
create_datasets()
return(result)
}
```

**Call It and Combine Your Data** Call the function you made two times to read in and parse the two `.csv` files mentioned so far. Be sure to call the new `value` column the same in both function calls.

Write **a single short function** that takes in the results of two calls to your wrapper function. The function should combine the tibbles appropriately (that is the two county level data sets get combined and the two non-county level data sets get combined). This can easily be done within your function using some calls to `dplyr::bind_rows()`. The function should then return two data sets as one object (in the same format as the input data sets as we will be combining this output with more calls to the wrapper function in a bit).

Call this function to combine the result of the two calls to the wrapper function.

```
tibble1 <- my_wrapper("https://www4.stat.ncsu.edu/~online/datasets/EDU01a.csv")

tibble2 <- my_wrapper("https://www4.stat.ncsu.edu/~online/datasets/EDU01b.csv")
```

### Writing a Generic Function for Summarizing

**First steps** We briefly discussed the ideas around object oriented programming and method dispatch. That is, we discussed why `plot(iris)` and `plot(exp)` give different types of plots. They look at the `class` of `iris` and the `class` of `exp` and the appropriate plotting function is called.

```
#Run these in your console
plot.function #what is used for a class = function
getS3method("plot","data.frame")  #what is used for a class = data frame
```

The generic function is `plot()`. Notice that it calls `UseMethod("plot")` and that is how the appropriate plotting function is determined.

```
plot
```

```
## function (x, y, ...)
## UseMethod("plot")
## <bytecode: 0x000001bb0c5911b0>
## <environment: namespace:base>
```

Great, well we have our own classes now (`county` and `state`). We can write our own custom `plot` function for these! We simply write our function as follows:

```
plot.state <- function(...){...}
```

For the `state` plotting method, let's write a function that plots the mean value of the statistic (enrollment for this data set) across the years for each `Division`. That is, on the x-axis we want the numeric year value, on the y-axis we want the mean of the statistic for each `Division` and numeric year. Also, we want to remove observations at the `ERROR` setting of `Division`.

```
plot.state <- function(df, var_name = "_your_default_value_"){
  #code to find the means for each division and year (think tidyverse)
  #Use get(var_name) to reference the var_name in your call to mean(): mean(get(var_name))
  #For this, think group_by along with summarize
  #remove the "ERROR" setting

  #create a line plot with ggplot with appropriate aes() assignments
}
```

Test out this function by running `plot(_class_state_df_here_, var_name = ...)`. (This doesn't need to go into the report here, just make sure it is working!)

For the class `county` we'll do a similar plotting function but with more flexibility. This function should allow the user to:

- specify the state of interest, giving a default value if not specified
- determine whether the 'top' or 'bottom' most counties should be looked at with a default for 'top'
- instruct how many of the 'top' or 'bottom' will be investigated with a default value of 5

Within your plot function you should:

- filter the data to only include data from the state specified
- find the overall mean of the statistic (use `get(var_name)` here as well) for each `Area_name` and sort those values from largest to smallest if 'top' is specified or smallest to largest if 'bottom' is specified
- obtain the top or bottom `x` number of `Area_name`s from the previous step where `x` is given by the user or the default
- filter the data for this state to only include the `Area_name`'s from the previous part (this is the data we'll use to plot)

Notice we aren't plotting the means here, but the actual statistic's value. Test out this function by running `plot(_class_county_df_here_)`. Run it a few more times specifying different input arguments. (This doesn't need to go into the report here, just make sure it is working!)

## Put it Together

The end of your report should have a section where you do the following:

- Run your data processing function on the two enrollment URLs given previously, specifying an appropriate name for the enrollment data column.
- Run your data combining function to put these into one object (with two data frames)
  - Use appropriate indexing (ex. [[1]]) to reference the correct data frame
- Use the plot function on the `state` data frame
- Use the plot function on the `county` data frame
  - Once specifying the state to be "NC", the group being the top, the number looked at being 20
  - Once specifying the state to be "SC", the group being the bottom, the number looked at being 7
  - Once without specifying anything (defaults used)
  - Once specifying the state to be "PA", the group being the top, the number looked at being 8

Lastly, read in another couple similar data sets and apply your functions!

- Run your data processing function on the four data sets at URLs given below:
  - https://www4.stat.ncsu.edu/~online/datasets/PST01a.csv
  - https://www4.stat.ncsu.edu/~online/datasets/PST01b.csv
  - https://www4.stat.ncsu.edu/~online/datasets/PST01c.csv
  - https://www4.stat.ncsu.edu/~online/datasets/PST01d.csv
- Run your data combining function (probably three times) to put these into one object (with two data frames)
- Use the plot function on the `state` data frame
- Use the plot function on the `county` data frame
  - Once specifying the state to be "CA", the group being the top, the number looked at being 15
  - Once specifying the state to be "TX", the group being the top, the number looked at being 4
  - Once without specifying anything (defaults used)
  - Once specifying the state to be "NY", the group being the top, the number looked at being 10

## Submission

For the submission link, you should upload give the link to your nicely rendered webpage!

## Rubric for Grading (total = 100 points)

| Item | Points | Notes |
|---|---|---|
| Data Processing Functions | 40 | Worth either 0, 5, 10, ... 40 |
| Combining Data Functions | 10 | Worth either 0, 5 or 10 |
| Generic Functions | 35 | Worth either 0, 5, ..., 35 |
| Putting it all together | 15 | Worth either 0, 5, 10, or 15 |

Notes on grading:

- For each item in the rubric, your grade will be lowered one level for each each error (syntax, logical, or other) in the code and for each required item that is missing or lacking a description.
- You should use Good Programming Practices when coding (see wolfware). If you do not follow GPP you can lose up to 50 points on the project.
- If the webpage doesn't work correctly you can lose up to 30 points.